ISO/IEC JTC 1/SC 22/WG 23 N 0330

Meeting #17 markup of Proposed revision of LAV in Ada annex

Date	25 March 2011
Contributed by	Secretary
Original file name	
Notes	Replaces N0311

I have shortened the vulnerability LAV in the Ada Annex considerably and removed all the redundant stuff about unsafe programming. Sorry that I did not do it in change mode to make the changes apparent. (retroactive comparison produced a stupid change version.)

Ada.23 Initialization of Variables [LAV]

Ada.23.1 Applicability to language

As in many languages, it is possible in Ada to make the mistake of using the value of an uninitialized variable. However, as described below, Ada prevents some of the most harmful possible effects of using the value.

The vulnerability does not exist for pointer variables (or constants). Pointer variables are initialized to null by default, and every dereference of a pointer is checked for a **null** value. ~~Therefore the vulnerability does not exist for pointer variables (or constants).~~

The ~~mandated checks (described elsewhere)~~checks mandated by the type system to ~~prevent memory corruption or operations on invalid values for given subtypes~~ apply to the use of uninitialized variables as well. Use of an out-of-bounds value in relevant contexts causes an exception, regardless of the origin of the faulty value. (See NZN regarding exception handling.) Thus, ~~no vulnerability exists beyond the~~the only remaining vulnerability is the potential use of a faulty but subtype-conformant value of an uninitialized variable, since it is technically indistinguishable from a value legitimately computed by the application.

For record types, default initializations may be specified as part of the type definition.

For controlled types (those descended from the language-defined type Controlled or Limited_Controlled), the user may also specify an Initialize procedure which is invoked on all default-initialized objects of the type.

The **pragma** Normalize_Scalars can be used to ensure that scalar variables are always initialized by the compiler in a repeatable fashion. This **pragma** is designed to initialize variables to an out-of-range value if there is one, to avoid hiding errors.

Lastly, the user can query the validity of a given value. The expression X'Valid yields true if the value of the scalar variable X conforms to the subtype of X and false otherwise. Thus, the user can protect against the use of out-of-bounds uninitialized or otherwise corrupted scalar values.

Ada.23.2 Guidance to language users

This vulnerability can be avoided or mitigated in Ada in the following ways:

- If the compiler has a mode that detects use before initialization, then this mode should be enabled and any such warnings should be treated as errors.
- Where appropriate, explicit initializations or default initializations can be specified.
- The pragma Normalize_Scalars can be used to ~~as for~~cause out-of-range default initializations for scalar variables.
- The ‘Valid attribute can be used to identify out-of-range values caused by the use of uninitialized variables, without incurring the raising of an exception.

~~One supposed mitigation~~Common advice that should be avoided is to perform a “junk initialization” of variables. Initializing a variable with an inappropriate default value such as zero can result in hiding underlying problems, because the compiler or other static analysis tools will then be unable to ~~identify use~~detect that the variable has been used prior to receiving a correctly computed value. ~~before correct initialization.~~